자바의 함수형 프로그래밍

3월 28일(목)까지 제출

제출: 아래 지정한 4개의 파일을 작성하고 이들을 zip 하여 upload 할 것. zip 파일의 이름은 자신의 이름으로 함 (예: 홍길동.zp)

1 Functional Interface

1.1 인터페이스에 Generics 적용하기 (제출 파일이름 : FunInterfaceHW.java)

강의자료 9쪽 Square 가 다음과 같은 조건을 만족하도록 그 파일을 수정하라.

- Square 를 generics 형태인 Square<T,R> 이 되도록 한다.
- isA 라는 람다 식을 정의한다. 이 람다 식은 한 개의 문자를 입력 받아, 이것이 'A' 이면 true 이고, 'B' 이면 false 가 출력되도록 코딩하라.

1.2 java.util.function.* 이용하기 (제출 파일이름 : MathHW.java)

수업시간에 실습했던 FunInterface2.java 파일의 수정을 완성한다. 원래 있었던 FunctonalInterface 를 사용하지 않고 import되는 패키지에서 미리 정해진 인터페이스을 적용한다. 결과적으로 MathOperation, GreetingService 인터페이스가 제거되고, operate 메소드도 제거되며, new 의 사용도 없어야 한다. 수정된 파일의 출력은 원래의 파일과 동일한 형태가 되도록 한다.

1.3 빈칸 채우기 (제출 파일이름: FunInterface2HW.java)

다음 프로그램은 자체적으로 정의한 Functional Interface와 패키지를 이용한 두 가지 경우가 혼합되어 있다. 밑줄친 빈 칸을 채워서, 프로그램의 수행결과 다음 내용이 출력되도록 프로그램을 완성하라.

출력: ————————————————————

```
25 25 25 120
false false false
100 , BiConsumer : 100
```

————————————————————-

위의 결과를 얻을 수 있도록 밑줄 친 빈칸을 채워서 코드를 완성하라.

```
import java.util.function.*;

@FunctionalInterface
interface Square {
    int calculate(int x);
}

@FunctionalInterface
```

```
interface Max {
    boolean isFirst(int x, int y);
}

class FunInterface {

  static UnaryOperator<Integer> fac = n -> n == 0    // static, recursive method
      ? 1
      : n * FunInterface.fac.apply(n-1);

  public static void main(String args[]) {

      // Unary Functions
      Square square = x -> x*x;
      ____________________ square2 = x -> x*x;
      UnaryOperator<Integer> square3 = x -> x*x;

      System.out.println(square.__________(5));
      System.out.println(square2.apply(5));
      System.out.println(square3._____(5));
      System.out.println(fac._____(5));

      // Binary Functions
      Max isBigger = (x,y) -> x > y;
      ______________________________ isBigger2 = (x,y) -> x > y;
      BiPredicate <Integer,Integer> isBigger3 = (x,y) -> x > y;

      // lambda definition with multiple statements
      BinaryOperator <Integer> smallerSquare = ((x,y) -> {
          Integer smaller = x > y ? y : x;
          return square2.apply(smaller);
      });

      // Consumer : no return value
      ________________________ smallerSquare2 = ((x,y) -> {
          Integer smaller = x > y ? y : x;
          System.out.println("BiConsumer : " + square2.apply(smaller));
      });

      System.out.println(isBigger._________(10,20));
      System.out.println(isBigger2.apply(10,20));
      System.out.println(isBigger3.____(10,20));

      System.out.println(smallerSquare._____(10,20));
      smallerSquare2.accept(10,20);
  }
}
```

2 배열로부터 List 생성

- 주어진 배열을 리스트로 변환할 때 `Arrays` 클래스의 `asList` 를 적용하는데, 이것을 사용하기 위해서는 `import java.util.Arrays;` 를 수행한다. 이때 생성되는 List는 그 원소의 수가 고정되므로

원소를 제거하거나 추가할 수 없다.

- `ArrayList` 클래스의 생성자(Constructor)는 Collection 타입의 인수들을 size 변경이 가능한 List 로 변환한다. 이 클래스를 이용하기 위해서는 `import java.util.ArrayList;` 를 수행한다.

2.1 실습

- 다음을 실습하여 run-time 에러의 발생 여부를 확인할 것.

```
import java.util.Arrays;     // for Arrays.asList. List of fixed-size
import java.util.ArrayList;  // for ArrayList  Resizable List
import java.util.List;

class T {
    public static void main(String[] args) {
        List<Integer> list1 = new ArrayList<Integer>(Arrays.asList(4,7,2,8,9));
        List<Integer> list2 = Arrays.asList(4,7,2,8,9);

        list1.add(10);
        // list2.add(10); // runtime error

        list1.forEach(System.out::println);
    }
```

3 ForEach

`Collection` 인터페이스의 수퍼클래스인 `Iterable` 인터페이스에는 `forEach` 메소드가 등록되어 있으며, 따라서 `Collection` 의 서브 클래스에서는 이 메소드를 구현하고 있다. 이 메소드는 다음과 같은 타입을 갖는다.

```
void forEach(Consumer action)
```

- `forEach` 는 `Consumer` 형태의 함수를 인수로 갖는다. 즉, 이 함수는 입력은 갖지만 return 값은 갖지 않는 형태로서, print 가 대표적인 경우이다. 이 함수는 Functional Interface 형태의 함수로서, lambda exression 이나 method reference 형태로 표현될 수 있다.

- `forEach` 는 `Collection` 형태의 객체(예를 들어, List)에 대해서, 각 원소에 인수에 주어진 함수를 적용한다. 즉, `forEach` 는 iteration의 기능을 내재하고 있으며, 따라서 `for` 등을 표현하지 않더라도 이에 상응하는 iteration이 발생한다.

- `forEach` 를 수행한 결과 return 되는 값은 없으며 (`void` 타입), 이러한 특징때문에 stream 의 마지막 연산으로도 이용될 수 있다.

3.1 문제 (제출 파일이름: `ForEachHW.java`)

- 다음 코드는 `list1` 에 속한 원소 중에 가장 큰 수를 찾고, 또한 이둘 중 짝수를 찾아서 `evens` 리스트에 참가하는 기능을 한다.

- 이 코드는 `for` 를 사용하는 경우와, `forEach` 를 사용하는 두 가지 경우로 표현한다. 그러나 `forEach` 의 람다 식을 사용하는 경우 `max` 를 계산할 수 없다. 왜 이것이 불가능 한지를 설명하라.

- `forEach` 에서 `evens` 를 구하는 것은 가능하다. 다음과 같은 결과 값이 출력되도록 밑줄 친 부분을 완성하라.

출력: ————————————————

```
9
[4, 2, 8, 6]
[4, 2, 8, 6]
```

————————————————————————-

```
import java.util.Arrays;
import java.util.ArrayList;
import java.util.List;

class ForEach {
public static void main(String args[]) {
    List<Integer> list1 = Arrays.asList(4,7,2,8,9,3,6);   // list1 is fixed.
    ArrayList<Integer> evens = new ArrayList<Integer>();  // list2 is resizable.

    Integer max = 0;
    for(Integer e : list1) {
        if (e > max)
            max = e;
        if (e % 2 == 0)
            evens.add(e);
    }
    System.out.println(max);
    System.out.println(evens);

    evens.clear();
    list1.forEach (e -> {               // lambda expression
        -------------------------;
        // if (e > max)
            //  max = e;      // No assignment. Why?
    });
    System.out.println(evens);
    }
}
```

4 Stream

4.1 Stream의 생성 (자바의 정석 슬라이드 24쪽, 강의자료 슬라이드 14)

- Collection으로부터 생성 : `list.stream()`
- 배열로부터 생성 : `Stream.of(...)` , `Arrays.stream(new String[] {....})`
- 정수의 range 로부터 생성 : `IntStream.range(1,5)`
- 난수(random number) 생성 : `new Random().ints()`

4.2 Stream의 중간연산 (자바의 정석 슬라이드 27쪽, 강의자료 12쪽)

- `skip, limit`
- `filter, distinct`

- sorted
- map

4.3 Stream 최종 연산

- void 타입 (복귀값이 없는 경우) : forEach
- 배열로 변환 : toArray
- 통계정보 관련 함수 : count, sum, average, max, min
- 누적 계산 : reduce
- Collectors 관련 메소드 : collect(Collectors.toList())

4.4 실습

다음 프로그램 코드를 수행해 보고 코드의 동작을 이해할 것.

```
import java.util.List;
import java.util.Arrays;
import java.util.stream.*;
import static java.util.stream.Collectors.*;

public class Lazy {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8);
        List<Integer> twoEvenSquares = numbers.stream()
            .filter(n -> {
                    System.out.println("filtering " + n);
                    return n % 2 == 0;
                })
           .map(n -> {
                    System.out.println("mapping " + n);
                    return n * n;
                })
           .limit(2)
           .collect(toList());

        System.out.println(twoEvenSquares);
    }
}
```